HARDWARE-ACCELERATED COMPUTATION OF RADIANCE TRANSFER COEFFICIENTS IN COMPUTER GRAPHICS

Technical Field

The invention relates generally to computer graphics techniques for rendering images of a modeled object with realistic lighting.

Background

Generating accurate depictions of complex scenes in interesting lighting environments is one of the primary goals in computer graphics. The general solution to this problem requires the solution of an integral equation that is difficult to solve even in non-interactive settings. In interactive graphics, short cuts are generally made by making simplifying assumptions of several properties of the scene; the materials are generally assumed to be simple, the lighting environment is either approximated with a small number of point and directional lights, or environment maps and transport complexity (i.e., how the light bounces around the scene, such as, inter-reflections, caustics and shadows) is only modeled in a limited way. For example, shadows may be computed for dynamic point lights, but not for environment maps.

Pre-computed Radiance Transfer

Sloan et al., "Graphics Image Rendering With Radiance Self-Transfer For Low-Frequency Lighting Environments", U.S. Patent Application No. 10/389,553, filed 3/14/2003, and published as Publication No. US-2003-0179197-A1 (the disclosure of which is hereby incorporated by reference) [hereafter "Sloan '553 application"], describes a technique called "precomputed radiance transfer" (PRT) that enables rigid objects to be illuminated in low frequency lighting environments with global effects like soft shadows and inter-reflections in real time. It achieves these results by running a lengthy pre-process that computes how light is transferred from a source environment to

exit radiance at a point. Previous methods for running this pre-process were designed to execute on the central processing unit (CPU) of a computer, and utilized the processing and memory resources of the CPU in a traditional manner.

Graphics Processing Unit

Computers commonly have a graphics adapter or graphics accelerator that contains a specialized microprocessor, generally known as a graphics co-processor or graphics processing unit (GPU). A GPU also can be integrated into the chip set contained on the motherboard of the computer. The GPU handles high-speed graphics-related processing to free the computer's central processing unit (CPU) for other tasks. Today's graphics adapters (e.g., various graphics adapter models available from NVIDIA and ATI Technologies, among others) feature GPUs that are specifically designed to render 3-dimensional (3D) graphics images and video at high frame rates, such as for use in computer games, video and other graphics intensive applications. Some CPUs for computers also include specialized instructions in their instruction sets that are designed for graphics-related processing (e.g., the MMX instructions in Intel Corporation microprocessors).

In past graphics adapters, the GPU generally provided fixed functionality for graphics processing operations. Application programs (e.g., a game with 3D graphics) would interact with the graphics adapter through a graphics application programming interface (API), such as Microsoft Corporation's DirectX®, and OpenGL® of Silicon Graphics, Inc. Through a graphics API, the application programs directed the GPU to execute its fixed graphics processing functions.

In its version 8, Microsoft DirectX® more recently introduced the concept of a programmable graphics shader for recent programmable graphics hardware. A shader is a program that executes on graphics hardware to perform graphics processing on a per pixel, or per vertex (or other graphics component or fragment) basis for 3D graphics rendering. DirectX® 8 included a language for writing shaders. DirectX® version 9 further introduced a high level programming language for shaders (called the High Level Shading Language or HLSL) to make it easier for application developers to create

shaders. The HLSL is syntactically similar to the well known C programming language. This makes it easier for programmers who are familiar with C to understand.

The architecture or design of current GPUs is optimized to execute particular graphics operations, but has limitations that prevent practical and efficient execution of others. The PRT preprocessing technique as previously implemented on the CPU includes operations that do not execute efficiently on the GPU. Accordingly, a direct mapping of the previous PRT preprocessing technique from the CPU onto GPU hardware would not be practical or efficient. For example, current GPUs do not practically or efficiently execute reduction operations, ray tracing, rasterization, as well as providing slow read-back paths, among others. In particular, the computations in the previous CPU-based PRT preprocessing technique that involve iterating over vertices and shooting rays do not map well to current GPU hardware.

Summary

A technique for hardware-accelerated computation of radiance transfer coefficients described herein is designed for practical and efficient execution on the GPU. The technique optimizes the radiance transfer computation to take into account the strengths and weaknesses of GPU architectures versus that of the CPU.

In one implementation, this hardware-accelerated radiance transfer computation technique is applied to precomputation of the PRT on the GPU. For efficient PRT pre-processing on the GPU, the computation is re-ordered relative to the previous PRT preprocessing technique on the CPU. In particular, the hardware-accelerated radiance transfer computation iterates over a sampling of directions about an object, as compared to iterating over points sampled over the object as in the previous PRT preprocessing technique. This enables the inner loop of the computation to be performed using a pixel shader as operations on a set of textures representing positions and normals of a set of sampling points over the object mapped into texture space. In this inner loop of the computation, a texture is computed containing the depth of the object for the currently iterated direction, the radiance transfer contribution for the direction at the position in the

texture is calculated, and the texture for the direction is accumulated into texture representing the final transfer coefficients.

Additional features and advantages of the invention will be made apparent from the following detailed description of embodiments that proceeds with reference to the accompanying drawings.

Brief Description Of The Drawings

Figure 1 is a drawing depicting the integration of radiance transfer for radiance-transfer precomputation.

Figure 2 is a flow diagram of a previous PRT preprocessing technique.

Figure 3 is a pseudo-code listing of the previous PRT preprocessing technique.

Figure 4 is a pseudo-code listing of a radiance transfer computation suitable for hardware-accelerated execution, such as on a GPU.

Figure 5 is a flow diagram of the hardware-accelerated radiance transfer computation of Figure 4.

Figure 6 is a flow diagram of a rendering pass within a computation loop in the hardware accelerated computation of radiance transfer coefficients of Figure 5.

Figure 7 is a code listing of a pixel shader for execution on a GPU to perform the update in an inner-loop of the hardware-accelerated radiance transfer computation of Figure 4.

Figure 8 is a code listing of a pixel shader for execution on a GPU to perform image rendering based on the pre-computed radiance transfer.

Figure 9 is a block diagram of a suitable computing environment for implementing the hardware-accelerated radiance transfer computation of Figure 1.

Detailed Description

The following description is directed to various implementations of techniques for hardware-accelerated computation of radiance transfer coefficients. The description presents an exemplary application of this technique to computer graphics image rendering based on pre-computed radiance transfer (PRT), such as described in the Sloan

'553 application. More particularly, an exemplary application of the technique evaluates the PRT preprocess for diffuse objects with shadows efficiently on the GPU. However, the hardware-accelerated computation of radiance transfer coefficients alternatively can be applied to other image rendering applications in computer graphics.

1. Precomputed Radiance Transfer Overview

As described more fully in the Sloan '553 application, PRT is a technique that enables rendering images of rigid objects illuminated in low frequency lighting environments with global effects like soft shadows and inter-reflections in real time. The technique achieves this by running a lengthy preprocess that computes how light is transferred from a source environment to exit radiance at a point. The following section summarizes the basis of this PRT technique.

For a diffuse object illuminated in distant lighting environment L, the reflected radiance at a point P on the surface is:

$$R_p(\vec{v}) = \frac{\rho_d}{\pi} \int_s L(s) V_p(s) H_N(s)\, ds \qquad (1)$$

Where V_p represent visibility; a binary function that is 1 in a given direction if a ray in that direction originating at the point can "see" the light source, and 0 otherwise. H_N represents the projected area (or cosine term) and the integration is over the hemisphere about the points normal. The diffuse reflectance (or albedo) of the surface is ρ_d and is generally a RGB color where each values is in between zero and one. The division by π maps irradiance (the integral) into exit radiance (what we see) and guarantees energy conservation (i.e., the amount of energy reflected is never greater then the amount of energy arriving at a point.)

With a point or directional light, the lighting environment is effectively a delta function, which turns the integral into a simple function evaluation – the cosine of the angle between the light and the normal if the direction is not in shadow or just zero if it is. Since the object is diffuse, the reflected radiance is the same in all directions and the

integral does not depend on the view direction. The key idea behind precomputed radiance transfer is to approximate the lighting environment using a set of basis functions over the sphere:

$$L(\vec{s}) \approx \sum_i l_i B_i(\vec{s}) \tag{1}$$

Where the B's are a set of basis functions and the l's are the coefficients corresponding to the optimal (in a least squares sense) projection of the lighting environment into the basis functions, that is they minimize:

$$\int (L(\vec{s}) - \sum l_i B_i(\vec{s}))^2 ds \tag{2}$$

If the basis functions are orthogonal this just amounts to integrating the lighting environment against the basis functions, while in general it is necessary to integrate against the duals of the basis functions.

Now substitute the approximation of the lighting environment into (1):

$$R_p(\vec{v}) \approx \frac{\rho_d}{\pi} \int_s \left(\sum_i l_i B_i(\vec{s}) \right) V_p(\vec{s}) H_N(\vec{s}) ds \tag{3}$$

And now recall two concepts from basic calculus: that the integral of a sum equals the sum of the integrals and that constants can be pulled outside of integrals. This allows us to reformulate (3) as follows:

$$R_p(\vec{v}) \approx \rho_d \sum_i l_i \int_s \frac{1}{\pi} B_i(\vec{s}) V_p(\vec{s}) H_N(\vec{s}) ds \tag{4}$$

The important thing to note about the above equation is that the integral only depends on the choice of basis functions, not on the value of the particular lighting

environment or the albedo of the surface. This means that if you precompute the integral for each basis function at every point on the object you are left with the following expression for reflected radiance:

$$R_p(\tilde{v}) \approx \rho_d \sum_i l_i t_{pi} \qquad (5)$$

A dot product between the global lighting coefficients and the spatially varying (through the index p) transfer vector scaled by the albedo is all that is required. One implementation of the hardware-accelerated radiance transfer technique described herein is efficient computation of t_{pi}.

2. Previous PRT Pre-Computation

With reference now to Figures 1-3, the Sloan '553 application presents a pre-process 200 (Figure 2) for computing the PRT of an object. In this pre-process, the radiance transfer including shadow effects is first computed in a first "shadow" pass 202, then the contribution to radiance transfer from interreflections is computed in one or more additional "interreflection" passes 204. In the shadow pass 202, the preprocess iterates over a set of points (e.g., vertices) on the object at 210. Then, in an inner loop at 211-213, the preprocess iterates over directions from the point (as depicted in Figure 1), integrating the shadow transfer at the point over the directions. In the interreflections pass(es) 204, the preprocess again iterates over the points on the object at 220, now integrating interreflection transfer at the respective point over the occluded directions in inner loop 221-223.

One example implementation of the hardware-accelerated radiance transfer technique described herein focuses on evaluating the PRT preprocess for diffuse objects with shadows efficiently on the GPU. In other words, the technique provides more efficient computation of the radiance transfer including shadows, which compares to the shadow pass 202 of the previous PRT preprocess 200. This previous preprocess for

precomputing the PRT with shadow effects can be expressed in pseudo-code 300 as shown in Figure 3.

For each point (P), this pseudo-code process 300 (Figure 3) iterates over a set of uniform directions on the unit sphere (as in Monte-Carlo integration), only adding the contribution for the basis functions in the directions that can see the light (again, as depicted in Figure 1). The contributions are just a numerical evaluation of the integral in equation (4) above, which is the basis functions times the cosine of the angle between the ray direction and the normal times the visibility function evaluated in the given direction. The normalization constant for monte-carlo integration is $\frac{4\pi}{N}$, where the surface area of the sphere is 4π and N is the number of samples. This gives the irradiance at each point. The process then divides by π to turn that into the radiance leaving the point. This resulting radiance transfer coefficient for the point is then stored as part of the PRT data to be used in rendering images of the object with shadow effects, as also described in the Sloan '553 application.

Unfortunately, the previous PRT precomputation as expressed in the flow chart of Figure 2 and the pseudo-code 300 of Figure 3 is not suitable for efficient, accelerated execution on graphics hardware. The previous PRT precomputation technique requires ray tracing, which while possible to do on graphics hardware is not very efficient. Further, the previous PRT precomputation requires an accumulation of disparate points in space, which also can not be done very efficiently on the GPU.

3. Hardware-Accelerated Radiance Transfer Computation

In accordance with one implementation of the hardware-accelerated computation of radiance transfer coefficients technique, the PRT preprocess is altered to become more suitable for hardware-accelerated execution on a GPU. This altered or hardware-accelerated version of the PRT preprocess can be expressed in pseudo-code 400 as shown in Figure 4. As compared to the previous PRT preprocess pseudo-code 300 of Figure 3, the order of the inner- and outer-loops of the hardware-accelerated PRT preprocess 400 are reversed to be more suitable for GPU execution. In particular, while the previous

PRT preprocess iterated over sample points on the object in the outer loop and over directions from the respective point in the inner loop, this hardware-accelerated PRT preprocess iterates over directions in the outer loop and points in the inner loop.

The hardware-accelerated PRT preprocess uses a couple of textures, which are defined as follows: object space position texture (G) is a texture that contains the position of each point mapped into texture space; and object space normal texture (N) is a texture with the same correspondence as above, but contains the surface normal at each sample instead of position. If the object has a parameterization, such parameterization can be used for mapping the position into texture space in the texture G. Otherwise, it is reasonable to simply pack the points or vertices into the texture in a coherent fashion.

With reference now to the flow chart depiction of the hardware-accelerated PRT preprocess 500 shown in Figure 5 (which corresponds to the pseudo-code 400 in Figure 4), the object space position (G) and object space normal (N) textures are first generated and initialized at 510. In one implementation, the object space position texture (G) is stored in RGB (red/green/blue) component form, using a 16-bit fixed or floating point number format. Alternative implementations can use different texture size and formats. In some implementations, the mapping of the position and normal into texture space can use a parameterization of the object, or a unique mapping of vertex to pixel. For example, when using PRT with textured objects, the objects can require unique texturing over the object surface. However, other implementations may use other mappings of the points into texture space, including arbitrary mappings that need not be a one-to-one parameterization. In some implementations, this unique point to texel mapping can be performed using a conventional texture synthesis algorithm (as described in, e.g., Turk, G., *Texture Synthesis on Surfaces*, SIGGRAPH 2001, 347-354), which can use a texture atlas for the mapping (e.g., to minimize distortion). For example, any mapping of the sphere to a square by a single cut will have a lot of distortion, but if the sphere is broken into patches the individual patches will have much less distortion. In deciding the number of patches, a trade-off is made between inefficiencies from packing the patches into the texture and distortion. If a texture atlas is used, a "gutter" region preferably is defined and filled with valid values, so that bi-linear interpolation can be used. In this

implementation of the hardware-accelerated PRT preprocess 500, the object is drawn once at a near depth and then 8 more times with single texel shifts and increased depths – effectively replicating the samples on the boundary into the gutters.

At 511, the hardware-accelerated PRT preprocess 500 builds a number N of textures to store the transfer coefficients, where N/4 is the number of basis functions used to represent the radiance transfer.

After initializing the textures, the hardware-accelerated PRT preprocess 500 iterates over a sampling of directions in a loop 512-515. In one implementation, the sampling of directions are generated as uniformly distributed points on a unit sphere, using a standard mapping from the unit square to the sphere and jittered sampling. These are the directions that are used to perform the numerical integration for the PRT precomputation.

In iterating over the direction in the loop 512-515, the hardware-accelerated PRT preprocess 500 performs the following basic computations 513-514 for radiance transfer with shadow (shadow pass): a texture is computed containing the depth of the object from that texture (a shadow z-buffer essentially) at 513 so as render the entire object (iterating over position effectively); the contribution for the given direction is computed in a texture; and finally this texture is accumulated into the final results at 514. The shadow Z buffer is a texture that stores the depth of the object in the direction (e.g., as a 16-bit fixed or floating point format). The shadow pass is fairly straightforward – an orthographic projection matrix is appended to a transformation that sets the view direction (the Z axis) to the current direction being computed. Care should be taken to generate as tight a frustum around the object as possible to maximize the texture's spatial resolution and precision. High precision textures should be used to store the depth (either 16 bit floating point or fixed point should suffice for most objects.)

Figure 6 shows the steps in this shadow pass inner loop 600 (corresponding to step 514 of Figure 5) in more detail. At 610, the shadow pass inner loop binds the camera depth, position and normal textures. The loop then samples position and normal textures using the respective position in the transfer coefficient texture at 611. At 612, the loop computes the cosine term H_N (from equation (4) above). The loop projects the

position into the orthographic camera at 613. The loop samples depth using the position (x,y), and compares with the computed depth to determine visibility (i.e., occlusion or shadowing) at 614. The loop then sums the contribution from this direction into the radiance coefficient texture. The contribution is determined by multiplying by the cosine term, and the result of the visibility comparison (1 or 0).

In one implementation, the bulk of this shadow pass inner loop can be computed in a pixel shader 700 (whose assembly language listing 700 is shown in Figure 7) on the GPU. In this pixel shader, simple geometry is drawn that rasterizes the corresponding UV coordinate for each pixel on the screen. A square can be used but a single triangle that encompasses the normalized device coordinates (NDC) is potentially more efficient. The vertex shader just maps from UV coordinates to NDC coordinates and emits the texture coordinates along with position. The transformation used for the pixel shader is identical to the one used in the vertex shader of the shadow pass but appended with a transformation from NDC coordinates to UV coordinates and biases the depth value to minimize "acne" that can occur with shadow depth buffer techniques.

For finally accumulating the contributions of the current direction into the radiance coefficients texture, it is unfortunately not possible to blend into floating point render targets with the current generation of graphics hardware. The hardware-accelerated PRT preprocess 500 overcomes this limitation in such GPUs by using three sets of buffers: a buffer (4 textures) that represents the most recent direction, a buffer that represents the previous frame's approximation of the integral, and a buffer that is two frames old. The hardware-accelerated PRT preprocess loads a further pixel shader that adds the current contribution into the most recent accumulation buffer, the roles of the accumulation buffers alternate every frame (e.g., always accumulate into texture buffer (uDir)&1 and read from (uDir+1)&1.) The pixel shader has two constants that multiply each of the terms – the previously accumulated results are blended with a zero on the first update and a one after, and the current frame is always updated with the normalization factor 4/uNumDirections, where uNumDirections is a constant that represents the number of directions sampled by the hardware-accelerated PRT preprocess.

4. Final Rendering

When the hardware-accelerated PRT preprocess is finished, the resulting PRT data can be used to render images of the object in a lighting environment as described in the Sloan '533 application. In one implementation, this image rendering with PRT produced by the above-described hardware-accelerated PRT preprocess can use a pixel shader (shown in the assembly code listing 800 of Figure 8) that computes the large dot products for the red, green and blue projection of the lighting environment. The contents of the texture could be transferred to a fixed point texture which can be filtered, such as with GPUs available from ATI Technologies, Inc. Care should be taken to maximize the precision when doing this – each channel can map its largest absolute value to one and then scale the corresponding coefficient of the lighting environment (which is always stored in high precision constant registers) by one over this scale factor. The projection coefficients of the lighting environment have to be rotated into the object space of the object. The sequence of muls/mads used in the pixel shader shown in listing 800 scales to larger dot products more efficiently than simply using dp4 instructions.

5. Alternative Implementations

There are several straightforward extensions and improvements that can be made to the hardware-accelerated PRT preprocess detailed above. Multiple directions can be done in a single pass by storing the four shadow z-buffers in the separate color channels of a single texture and having a longer pixel shader. If the directions are coherent, this should have reasonable cache performance, and would cut down on the number of accumulation steps that are necessary. This could also alleviate some of the precision issues associated with the accumulation step.

Further, ID buffers, percentage closer filtering, and angle dependent offsets can be used in the hardware-accelerated PRT preprocess to make the shadow depth buffering phase more accurate.

Finally, the normal texture can be analyzed and slightly more complex geometry can be used based on clustering normals. Only subregions that can have non-zero dot products with the current direction would have to be rasterized. More complex

precomputed data structures also can be used to handle regions that can be conservatively classified as either completely shadowed or unshadowed from a given direction.

6. Computing Environment

The above described technique for hardware-acceleration of radiance transfer coefficients computation (such as, the above-detailed hardware-accelerated PRT preprocess 500 (Figure 5)) can be implemented on any of a variety of computing devices and environments that include hardware graphics accelerators (e.g., GPUs), including computers of various form factors (personal, workstation, server, handheld, laptop, tablet, or other mobile), distributed computing networks, and Web services, as a few general examples.

Figure 9 illustrates a generalized example of a suitable computing environment 900 in which the described techniques can be implemented. The computing environment 900 is not intended to suggest any limitation as to scope of use or functionality of the invention, as the present invention may be implemented in diverse general-purpose or special-purpose computing environments.

With reference to Figure 9, the computing environment 900 includes at least one processing unit 910 and memory 920, as well as the graphics processing unit 915. In Figure 9, this most basic configuration 930 is included within a dashed line. The processing unit 910 executes computer-executable instructions and may be a real or a virtual processor. In a multi-processing system, multiple processing units execute computer-executable instructions to increase processing power. The memory 920 may be volatile memory (e.g., registers, cache, RAM), non-volatile memory (e.g., ROM, EEPROM, flash memory, etc.), or some combination of the two. The memory 920 stores software 980 implementing the hardware-accelerated graphics transfer coefficients computation technique. The graphics processing unit 915 is a specialized microprocessor that handles high-speed graphics-related processing, such as may be provided on a graphics adapter or integrated into the chip set contained on the motherboard of the computer (e.g., various graphics adapter models currently available from NVIDIA and

ATI Technologies, among others). Preferably, the graphics processing unit 915 is programmable, so as to support executing pixel shaders.

A computing environment may have additional features. For example, the computing environment 900 includes storage 940, one or more input devices 950, one or more output devices 960, and one or more communication connections 970. An interconnection mechanism (not shown) such as a bus, controller, or network interconnects the components of the computing environment 900. Typically, operating system software (not shown) provides an operating environment for other software executing in the computing environment 900, and coordinates activities of the components of the computing environment 900.

The storage 940 may be removable or non-removable, and includes magnetic disks, magnetic tapes or cassettes, CD-ROMs, CD-RWs, DVDs, or any other medium which can be used to store information and which can be accessed within the computing environment 900. The storage 940 stores instructions for the device connectivity and networking software 980.

The input device(s) 950 (e.g., for devices operating as a control point in the device connectivity architecture 100) may be a touch input device such as a keyboard, mouse, pen, or trackball, a voice input device, a scanning device, or another device that provides input to the computing environment 900. For audio, the input device(s) 950 may be a sound card or similar device that accepts audio input in analog or digital form, or a CD-ROM reader that provides audio samples to the computing environment. The output device(s) 960 may be a display, printer, speaker, CD-writer, or another device that provides output from the computing environment 900.

The communication connection(s) 970 enable communication over a communication medium to another computing entity. The communication medium conveys information such as computer-executable instructions, audio/video or other media information, or other data in a modulated data signal. A modulated data signal is a signal that has one or more of its characteristics set or changed in such a manner as to encode information in the signal. By way of example, and not limitation, communication

media include wired or wireless techniques implemented with an electrical, optical, RF, infrared, acoustic, or other carrier.

The hardware-accelerated radiance transfer coefficients computation techniques herein can be described in the general context of computer-readable media. Computer-readable media are any available media that can be accessed within a computing environment. By way of example, and not limitation, with the computing environment 900, computer-readable media include memory 920, storage 940, communication media, and combinations of any of the above.

The techniques herein can be described in the general context of computer-executable instructions, such as those included in program modules, being executed in a computing environment on a target real or virtual processor. Generally, program modules include routines, programs, libraries, objects, classes, components, data structures, etc. that perform particular tasks or implement particular abstract data types. The functionality of the program modules may be combined or split between program modules as desired in various embodiments. Computer-executable instructions for program modules may be executed within a local or distributed computing environment.

For the sake of presentation, the detailed description uses terms like "determine," "generate," "adjust," and "apply" to describe computer operations in a computing environment. These terms are high-level abstractions for operations performed by a computer, and should not be confused with acts performed by a human being. The actual computer operations corresponding to these terms vary depending on implementation.

In view of the many possible embodiments to which the principles of our invention may be applied, we claim as our invention all such embodiments as may come within the scope and spirit of the following claims and equivalents thereto.